流体の流れ場のインタラクティブなデザイン
Interactive Design of Flow Field

佐藤 周平[†] 土橋 宜典[†**] 岩崎 慶[*] 山本 強[†] 落合 啓之[‡**]

Syuhei SATO[†1] Yoshinori DOBASHI[†**] Kei IWASAKI[*] Tsuyoshi YAMAMOTO[†] Hiroyuki OCHIAI[‡**]

† 北海道大学 † Hokkaido University * 和歌山大学 * Wakayama University

‡ 九州大学 ‡ Kyushu University ** JST CREST

E-mail: sato@ime.ist.hokudai.ac.jp[1]

1. はじめに

近年，コンピュータグラフィックス(CG)において煙や水などの流体現象を流体解析によりシミュレーションする研究が盛んに行われている[9]．また流体現象は，映画やゲームなどの映像において重要な要素の 1 つであり，様々な場面で利用されている．これらの映像制作では，リアルかつユーザの制約に従うような流体映像の生成が求められる．しかし，流体シミュレーションは非常に計算コストが高い．さらに，所望の流体アニメーションを作成するには，多くのパラメータを調整しなければならない．これは，非常に煩雑な作業である．

この問題を解決するために，数多くの手法が提案されている．その中でも所望の流体映像を得ることを目的とした研究として，流体シミュレーションの制御手法があげられる．これらの手法では，流体シミュレーションに外力を付加することで所望の形状として流体を表現する[2][3]．また，目標とする形状と現在の流体形状との差から，シミュレーションパラメータを調整する手法も存在する[11]．これらの手法により，所望の形状に制御された流体のアニメーションを作成することができる．しかし，コントロールパラメータの調整が必要であり，煩雑な作業となってしまう．

本稿では，上記の問題を解決するために，ユーザがリアルな流体の流れ場をインタラクティブにデザインできる手法を提案する．写実的な挙動を保ちつつ，ユーザ所望の流れを生成するために，ユーザ指定の制約と Navier-Stokes 方程式を考慮した目的関数を最小化することで流れ場を計算する．また，流れ場をラプラシアン固有関数[1]の線形和で表現することで，この最小化問題を効率的に解く．これにより，生成される流れ場は常に非圧縮性の法則を満たすことができる．提案法では，上記の最小化問題を解く際の行列演算を GPU 実装により並列化することで，インタラクティブに流れ場を編集することができる．

提案法の特徴を以下に示す．

- ユーザの指定する流れの場と Navier-Stokes 方程式により計算された流れ場を目的関数とした最小化問題を解くことで，所望のリアルな流れ場を生成する．

- 基底関数としてラプラシアン固有関数を使用する．ラプラシアン固有関数の性質から，提案法により生成される流れ場は常に非圧縮性の制約を満たす．

- 3 次元の流れ場を 2 次元の基底関数により表現することで，メモリ容量を大幅に削減する．特にシミュレーション空間の形が立方体であれば，基底関数は一組でよい．

2. 関連研究

Stam は Navier-Stokes 方程式の安定的な解法を提案し，CG において実用的な流体シミュレーション手法を開発した[8]．Stam の手法以降，様々な流体現象を対象とした数多くの解析手法が提案されている．それらの手法の詳細は[9] にまとめられている．しかし，流体シミュレーションは計算コストが非常に高く，結果を生成するまでに多大な時間がかかってしまう．また，所望の流体の動きを作成するためにはシミュレーションパラメータを試行錯誤的に調整しなければならない．

流体シミュレーションの制御に関する研究がいくつか存在する[2][3]．これらの手法では，流体をユーザの指定した通りに制御できる．しかし，制御パラメータや目標形状などを調整する必要があり，煩雑な作業を必要とする．近年では，低解像度で所望の流体の動きを作成し，それを高解像度化することで，所望のリアルな映像を生成する手法が提案されている[4][5][6]．これらの手法では，低解像度で作成さ

れた所望の動きを保ったまま，流体の詳細な動きを付加することができる．所望の動きを低解像度で作成できるため，効率的に映像の生成ができる．しかし，低解像度における所望の動きの生成には，依然としてパラメータの調整が必要であり，流体の動きをインタラクティブにデザインするような枠組みは存在しない．本研究では，流れ場のインタラクティブなデザインに焦点を当てる．

3. 提案手法の概要

図 1 に提案法の概要を示す．図 1 の D はディレイバッファであり，タイムステップ t_n に入力された速度場 $\mathbf{v}$ を保存して 1 タイムステップ後に出力する．まず，所望の流れ場を生成するために，ユーザは任意の場所に制御点を配置する（図 1:青球）．そして，制御点位置の速度をインタラクティブに指定する（図 1：赤矢印）．ここで，タイムステップ t_n における l 番目の制御点の速度を $\mathbf{v}_u(\mathbf{x}_l,t_n)(l = 0, 1, \ldots, N_u\text{-}1)$ とする．$\mathbf{x}_l$ は l 番目の制御点の位置，N_u は制御点の数を示す．本手法では，最小化問題を解くことで，非圧縮性かつ非粘性の流れ場 $\mathbf{v}(\mathbf{x},t_n)$ を計算する．この計算には，入力として $\mathbf{v}_u(\mathbf{x}_l,t_n)$ および $\mathbf{v}_{ns}(\mathbf{x},t_n)$ を用いる．$\mathbf{v}_{ns}(\mathbf{x},t_n)$ は 1 ステップ前に提案法により得られた速度場 $\mathbf{v}(\mathbf{x},t_{n-1})$ を用いて，非圧縮非粘性 Navier-Stokes 方程式を解くことで得られる（図 1 参照）．これらの速度場は，最小化問題を解く際の制約として用いられる．$\mathbf{v}(\mathbf{x},t_n)$ の計算後，煙の密度などのスカラー量を移流させ，ボリュームレンダリングにより可視化する（図 1 参照）．そして可視化された結果から，所望の効果が得られるよう，ユーザは制約を修正する．

3.1 最小化問題の定義

提案法では，流れ場 $\mathbf{v}(\mathbf{x},t_n)$ を基底関数 $\mathbf{\Phi}_i$ の線形和により次式のように表現する．

$$\mathbf{v}(\mathbf{x},t_n)=\sum_{i=0}^{N-1} w_i(t_n)\mathbf{\Phi}_i(\mathbf{x}) \tag{1}$$

ここで，$w_i(t_n)$ は i 番目の基底関数に対する係数であり，N は基底関数の数である．本手法では，基底関数としてラプラシアン固有関数を用いる[1]．これにより，生成される速度場は常に非圧縮性条件を満足する．本手法では，次式に示すように 2 つの誤差関数の和を最小化することで，リアルかつ所望の動きの流れが生成されるような係数ベクトル $\mathbf{w}(t_n)$ $(= (w_0(t_n), w_1(t_n), \ldots, w_{N-1}(t_n)))$ を算出する．

$$\arg\min_{\mathbf{w}(t_n)}(E_{usr}(t_n)+\alpha E_{ns}(t_n)) \tag{2}$$

ここで，α は2つの誤差関数の影響を調整するために使われるユーザ指定の係数である．

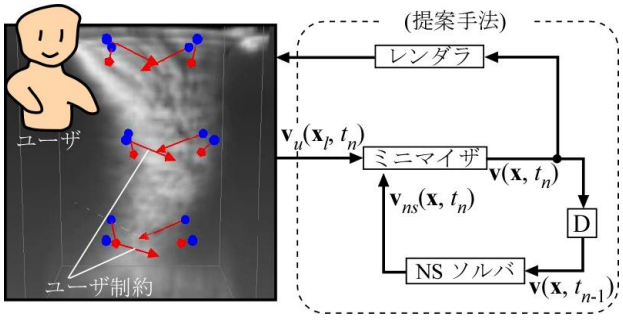

図 1：提案法の概要

$E_{usr}(t_n)$ は，制御点での速度と結果の流れ場との差分を表し，次式により定義される．

$$E_{usr}(t_n)=\sum_{l=0}^{N_u-1}|\mathbf{v}_u(\mathbf{x}_l,t_n)-\sum_{i=0}^{N-1} w_i(t_n)\mathbf{\Phi}_i(\mathbf{x}_l)|^2 \tag{3}$$

$E_{ns}(t_n)$ は，流れ場と非圧縮非粘性 Navier-Stokes 方程式との差分を表す．$E_{ns}(t_n)$ の定義を説明するために，まず，Navier-Stokes 方程式を以下に示す．

$$\dot{\mathbf{u}}=-(\mathbf{u}\cdot\nabla)\mathbf{u}-\frac{1}{\rho}\nabla p+\mathbf{f} \tag{4}$$

$$\nabla\cdot\mathbf{u}=0 \tag{5}$$

ここで，$\mathbf{u}$ は速度場，ρ は流体の密度，p は圧力，$\mathbf{f}$ は外力を示す．本手法ではラプラシアン固有関数を用いるため，連続の式（式(5)）は常に満たされる．そのため，$E_{ns}(t_n)$ には式(4)のみ用いる．次に，提案法では基底関数としてラプラシアン固有関数を用いるため，生成される速度場は常に非圧縮性の法則を満たす．非圧縮性 Navier-Stokes 方程式において，圧力項は式(5)を満足するために計算されるため，常に非圧縮性を満たすことができる本手法では式(4)の圧力項を解く必要がない．そのため，式(4)から圧力項を除き以下のようにする．

$$\dot{\mathbf{u}}=-(\mathbf{u}\cdot\nabla)\mathbf{u}+\mathbf{f} \tag{6}$$

そして，上式を時間に関して離散化し次式を得る．

$$\mathbf{u}(t_n)=\mathbf{u}(t_{n-1})+\Delta t\{-(\mathbf{u}(t_{n-1})\cdot\nabla)\mathbf{u}(t_{n-1})+\mathbf{f}\} \tag{7}$$

最後に，式(7)および，タイムステップ t_{n-1} における流れ場 $\mathbf{v}$ を用いて，$E_{ns}(t_n)$ は次式により定義される．

$$E_{ns}(t_n)=\|\mathbf{v}_{ns}(\mathbf{x},t_n)-\sum_{i=0}^{N-1} w_i(t_n)\mathbf{\Phi}_i(\mathbf{x})\|^2 \tag{8}$$

式(2)の最小化問題を $E_{usr}(t_n)$ のみを制約として解く場合，制御点の数が基底関数の数よりも大幅に少ないと適切に計算することができない．このような問題に対しては，正則化手法が一般的に用いられる．提案法では，$E_{ns}(t_n)$ が正則化項として機能する．そのため，最小化問題を安定に解くことができ，α の調整により正則化の度合いを制御できる．

3.2 最小化問題の解法

式(2)の最小化問題の解法について説明する．係数

$w_i(t_n)$に関して式(2)の微分をとることで,次の行列方程式を得る.

$$(\mathbf{A}_u + \alpha\mathbf{A}_{ns})\mathbf{w}(t_n) = \mathbf{c}_u(t_n) + \alpha\mathbf{c}_{ns}(t_n) \quad (9)$$

ここで，$\mathbf{A}_u$および$\mathbf{A}_{ns}$は$N\times N$の行列であり，$\mathbf{w}(t_n)$, $\mathbf{c}_u(t_n)$，$\mathbf{c}_{ns}(t_n)$はN次の列ベクトルである．$\mathbf{A}_u$および$\mathbf{c}_u(t_n)$はE_{usr}に関するものであり，$\mathbf{A}_{ns}$および$\mathbf{c}_{ns}(t_n)$はE_{ns}に関するものである．$\mathbf{A}_u$のi行j列目の要素$a_{ij}{}^u$と$\mathbf{c}_u(t_n)$のi番目の要素$c_i{}^u$はそれぞれ以下のようになる.

$$a_{ij}^u = \sum_{l=0}^{N_u-1} \mathbf{\Phi}_i(\mathbf{x}_l)\cdot\mathbf{\Phi}_j(\mathbf{x}_l)$$

$$c_i^u = \sum_{l=0}^{N_u-1} \mathbf{v}_u(\mathbf{x}_l,t_n)\cdot\mathbf{\Phi}_i(\mathbf{x}_l)$$

ここで・は2つのベクトル間の内積を表す．次に，$\mathbf{A}_{ns}$のi行j列目の要素$a_{ij}{}^{ns}$と$\mathbf{c}_{ns}(t_n)$のi番目の要素$c_i{}^{ns}$はそれぞれ以下のようになる.

$$a_{ij}^{ns} = \mathbf{\Phi}_i(\mathbf{x})\cdot\mathbf{\Phi}_j(\mathbf{x})$$

$$c_i^{ns} = \mathbf{v}_{ns}(\mathbf{x},t_n)\cdot\mathbf{\Phi}_i(\mathbf{x})$$

ここで，$\mathbf{\Phi}_i$は直交な基底関数であるため，$\mathbf{A}_{ns}$において$i\neq j$の成分では$a_{ij}{}^{ns}$の値が0となる.

式(9)の行列式を解くことで，係数ベクトル$\mathbf{w}(t_n)$が得られる．本稿では，この行列式を効率的に解くために，LU分解を用いる[7]．まず，行列$(\mathbf{A}_u+\alpha\mathbf{A}_{ns})$のLU分解を計算し，その分解された行列を用いて係数ベクトル$\mathbf{w}(t_n)$を効率的に算出する．しかし，この分解を毎ステップ計算する必要はない．行列$(\mathbf{A}_u+\alpha\mathbf{A}_{ns})$が変化するのは，$\mathbf{v}_u$において制御点の位置が変わる場合のみである.そのため,制御点の追加，削除および移動もしくは係数αの値を変更した場合のみ$\mathbf{A}_u$とLU分解を再計算する．これにより，行列$(\mathbf{A}_u+\alpha\mathbf{A}_{ns})$について逆行列を求める場合に比べ，高速に計算が可能である.

3.3 Radial Basis Functionsを用いた効率的な解法

前節で説明した方法では，制御点を更新する毎に$\mathbf{A}_u$とLU分解の計算が必要であり，3次元のシミュレーション空間において多くの制御点を指定する場合，高速に計算することが難しい．また，複雑な流れ場を生成する場合，必要な基底関数の数Nが増加する．LU分解の計算コストはN^3のオーダーに比例するため，この場合も計算時間が増加してしまう．これは特に3次元流れ場の編集において特に重大であり，インタラクティブな編集の実現における問題となる．そこで，本節では，この問題を解決するため，Radial Basis Function（RBF）を用いた効率的な解法について説明する.

基本的な考え方は,空間的に連続な流れ場$\mathbf{v}'_u(\mathbf{x},t_n)$を定義するために，RBFを用いてユーザの指定した速度場$\mathbf{v}_u(\mathbf{x},t_n)$を以下の式を用いて補間することである.

$$\mathbf{v}'_u(\mathbf{x},t_n) = \sum_{l=0}^{N_u-1} b_l(t_n)\cdot\phi(|\mathbf{x}-\mathbf{x}_l|) \quad (10)$$

ここで，ϕはRBFであり，$b_l(t_n)$は以下の等式を解くことで得られる.

$$\sum_{l=0}^{N_u-1} b_l(t_n)\cdot\phi(|\mathbf{x}-\mathbf{x}_l|) = \mathbf{v}_u(\mathbf{x}_m,t_n)$$

ここで，m=0,1,…,N_n-1である．ϕにはガウス基底関数：$\phi(r)$=exp(-(σr)^2)を用いる．ここで，σはユーザ指定の定数である.

次に$\mathbf{v}'_u$を用いて最小化問題を解く．$\mathbf{v}'_u$は全ての格子点で定義されるため,$\mathbf{A}_u$と$\mathbf{c}_u$の要素はそれぞれ以下のようになる.

$$a_{ij}^u = \mathbf{\Phi}_i(x)\cdot\mathbf{\Phi}_j(x)$$

$$c_i^u = \mathbf{v}'_u(x,t_n)\cdot\mathbf{\Phi}_i(x)$$

$\mathbf{\Phi}_i$は直交な基底関数であるため，上の式において計算される$a^u{}_{ij}$の$i\neq j$成分は0となる．そのため，$\mathbf{A}_u$および$\mathbf{A}_{ns}$は対角行列となり，その対角行列要素は前計算が可能である．そして，係数$w(t_n)$は以下の式で計算される.

$$w_i(t_n) = \left(c_i^u + \alpha c_i^{ns}\right)\Big/\left(a_{ii}^u + \alpha a_{ii}^{ns}\right) \quad (11)$$

上記のアプローチにより，ランタイムにおいて$\mathbf{A}_u$および$\mathbf{A}_{ns}$の計算が必要ないため，係数ベクトル$\mathbf{w}(t_n)$を非常に効率的に計算することができる．同様にLU分解もランタイムでの計算が必要ない．したがって，ランタイムでは，$\mathbf{v}'_u(\mathbf{x},t_n)$および$\mathbf{v}_{ns}(\mathbf{x},t_n)$を$\mathbf{\Phi}_i$へ投影する処理とその結果の係数の重みつき平均の計算のみ必要となる．このアプローチは制約となる速度を疎に設定した場合に特に効果的である．このアプローチにおいて起こりうる障害は、最終的な流れ場がRBFによって計算される値に近づく傾向があることである．しかし，我々の実験では，第5節に示すように満足のいく結果を得ている.

4. 基底関数

前述したとおり，本手法では基底関数としてラプラシアン固有関数を用いる．2次元および3次元のどちらに対しても，以下の式で定義される，矩形領域$[0,0]\times[s_x,s_y]$における2次元のラプラシアン固有関数$\mathbf{\Phi}_k$= ($f_{k1,k2}(x,y)$, $g_{k1,k2}(x,y)$)を用いる[1].

$$f_{k_1,k_2}(x,y) = \frac{1}{\lambda}k_2\sin(k_2\pi x/s_x)\cos(k_2\pi y/s_y) \quad (12)$$

$$g_{k_1,k_2}(x,y) = \frac{-1}{\lambda}k_1\cos(k_1\pi x/s_x)\sin(k_2\pi y/s_y) \quad (13)$$

ここで，k_1,k_2は波数，λは固有値でありλ=$k_1{}^2$+$k_2{}^2$として定義される．最小化問題を効率的に解くため，$\mathbf{\Phi}_k$は格子上で前計算され，行列$\mathbf{A}_u$，$\mathbf{A}_{ns}$とベクトル$\mathbf{c}_u$，$\mathbf{c}_{ns}$の要素を計算するために使用される.

3次元の基底関数は存在する[1]が，これらの関数

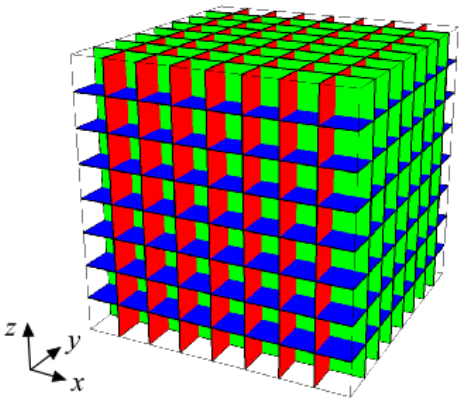

図 2：平面による分割

は我々の目的には適さない．3 次元の流れ場をデザインするためには十分な数の基底関数が必要である．本稿では 512 個の基底関数で試したが，ユーザの所望する流れ場のデザインを可能にするには十分ではなかった．さらに，3 次元の格子でサンプリングした 3 次元の基底関数のための記憶容量は非常に大きくなってしまう．これらの基底関数は高速に計算できるが，これはコンピュータの計算速度を低下させる．これらの問題を解決するために，我々は 3 次元の流れ場についても，上述した 2 次元の基底関数を使用することを提案する．

3 次元空間に対して，2 次元の基底関数を適用可能とするため，シミュレーション空間を xy（図 2：青平面），yz（図 2：赤平面），zx（図 2：緑平面）の 3 つの平面に分割する．そして，各平面において 2 次元の流れ場 $\mathbf{v}_{xy}$，$\mathbf{v}_{yz}$，$\mathbf{v}_{zx}$ を 3 節の方法を用いて算出する．これら 3 平面における 2 次元の速度場を用いて，格子点(p, q, r)における 3 次元の流れ場 $\mathbf{v}$ を以下の式により定義する．

$$\mathbf{v}_{yz}(p)=\sum_{k_1,k_2} w_{yz}(p)(0, f_{k_1,k_2}(y,z), g_{k_1,k_2}(y,z))$$

$$\mathbf{v}_{zx}(q)=\sum_{k_1,k_2} w_{zx}(p)(f_{k_1,k_2}(x,z),0, g_{k_1,k_2}(x,z))$$

$$\mathbf{v}_{xy}(r)=\sum_{k_1,k_2} w_{xy}(r)(f_{k_1,k_2}(x,y), g_{k_1,k_2}(x,y),0)$$

ここで，w_{yz}，w_{zx}，w_{xy} は各平面での流れ場を表現する係数である．格子点(p,q,r)での 3 次元の流れ $\mathbf{v}$ はこれら 3 つの 2 次元の流れ場の組み合わせで求められる．

$$\mathbf{v}(p,q,r)=\frac{1}{2}(\mathbf{v}_{xy}(p)+\mathbf{v}_{yz}(q)+\mathbf{v}_{zx}(r)) \qquad (14)$$

このように計算された流れ場は非圧縮性条件を満たす．同じ大きさの平面では同一の基底関数を用いることができるため，格子点においてサンプリングされた基底関数のための記憶容量を大幅に節約することができる．シミュレーション空間が立方体の場合，全ての平面に対して同じ基底関数を用いることができる．加えて提案法では、ユーザは極めて自由度の高い編集が可能である．しかしながら，ユーザが指定する速度場は制御点の存在する平面上でのみ制約を行っている．他の面にはユーザの意図は反映されないため，所望する流れ場の作成は難しい．この問題は 3.3 節に述べた方法により解決する．

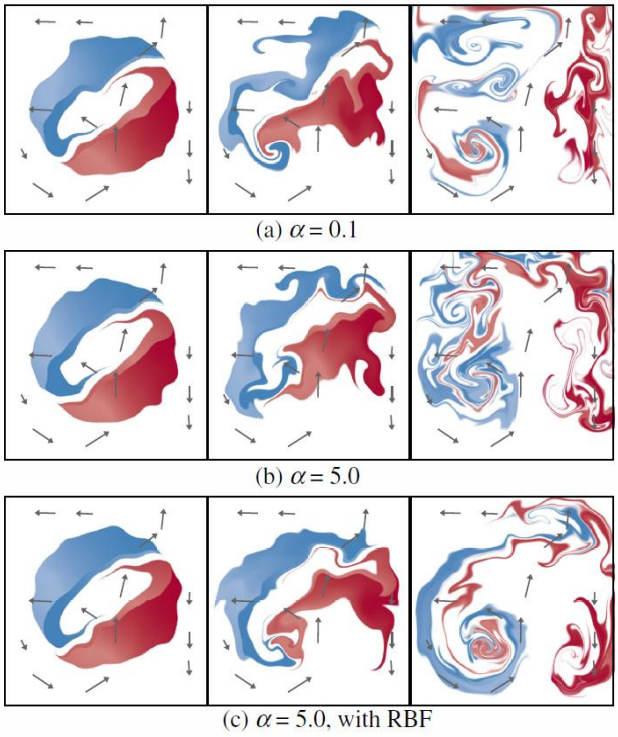

図 3：画像を移流させた例

5. 実験結果

提案法を用いて作成した適用例を図 3，4 および 5 に示す．実験環境は，CPU が Intel Core i7 2600K（メモリ 8GB），GPU が NVIDIA GeForce GTX680 の PC である．2 次元の結果（図 3）は，3.2 節の方法を用いている．3 次元に関しては，3.3 節の RBF を用いた方法を使用している．なお，3 次元の例については，従来法を用いて詳細を付加している．

図 3 は，2 次元の結果であり，提案法により得られた速度場に沿ってテクスチャ画像を移流させている．この例では，advection texture method を用いてテクスチャ座標を移流させている．ユーザにより指定された制約の速度場は灰色の矢印で示している．また，全ての結果において，基底数は 256 個，格子数は 32x32，1 ステップの計算時間は 15ms である．図 3(a)，(b)は，3.2 節の方法を用いて作成した結果であり，それぞれα=0.1，α=5.0に設定している．一

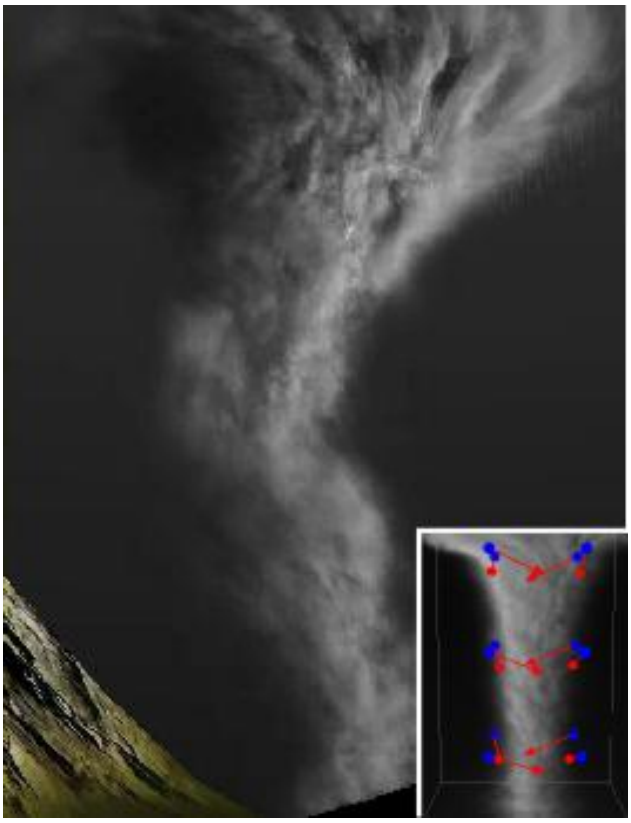

図 4: 竜巻の例

方，図3(c)は，3.3節の方法を用いて生成した例である．結果からわかるとおり，αを大きくするほど自然な流体の動きとなるが，ユーザ制約の影響は小さくなる．対称的に，RBF を用いた場合，流れはより大きく制御される．

図 4 は，提案法により竜巻のアニメーションを生成し，最終的なレンダリングまで行った例である．この例では，詳細形状を付加するために，手続き的に生成されたノイズを格納した 3 次元テクスチャおよび advection texture technique [10]を使用している．また，図右下は指定した制御点（青点）とその速度（赤矢印）を示しており，基底数は各スライスとも 256，格子数は 32x32x48，計算時間は 80ms である．指定した制約どおりの結果が生成されているのがわかる．

最後に図 5 は，3 次元の炎の例である．図 5(a)は，円形に渦巻く炎の例であり，シミュレーション空間の底に 5 つの球状の熱源を配置し，シミュレーションを行っている．図 5(b)は，ユーザにより指定されたパスに従って炎が流れるアニメーションを生成した例である．この例では，炎の熱源をシミュレーション空間の左端に配置している．図 5(a)では左上に，図 5(b)では左下に，それぞれの例で指定した制御点を示す．図 5(a)，(b)ともに，基底数 256 個，計算時

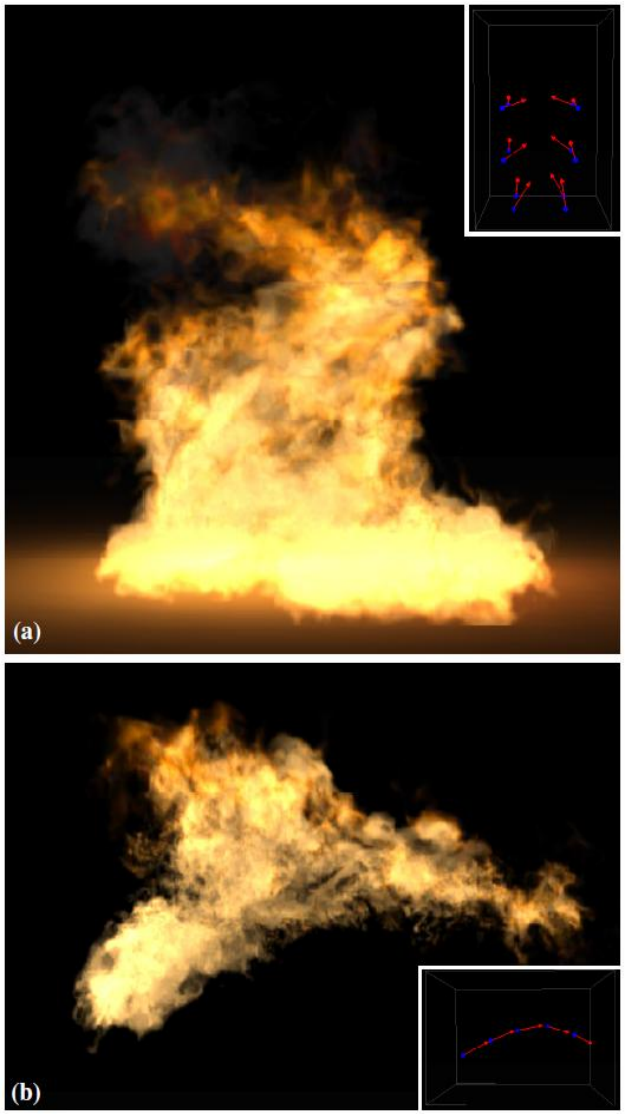

図 5：渦巻く炎とパスに従う炎の例

間は 80ms である．格子数は，図 5(a)が 32x32x48，図 5(b)が 48x32x32 である．また，詳細付加のために wavelet turbulence[6]の手法を適用した後，レンダリングを行っている．このように浮力を伴う炎であっても，ユーザの指定に沿った結果が生成できているのがわかる．

6. まとめと今後の課題

本稿では，非圧縮非粘性の流体の流れ場をインタラクティブにデザイン可能な手法を提案した．最小化問題を解くことで，ユーザ指定の制約と Navier-Stokes 方程式の両方を満たす流れ場を生成可能である．また，基底関数としてラプラシアン固有関数を用いているため，常に非圧縮性の制約を満たした流れ場を生成できる．さらに，2 次元の基底関数を用いて 3 次元の流れ場を表現する方法も提案し

た.

今後の課題としては，障害物を考慮した流れ場の生成があげられる．現状では，提案法により生成される速度場は障害物の影響を考慮できない.しかし，Witt らの手法[1]により解決できるため，今後本手法に組み込む予定である.

参考文献

[1] T. D. Witt, C. Lessig, E. Fiume 2012, Fluid simulation using laplacian eigenfunctions, *ACM Transactions on Graphics 31*, 1, Article 10

[2] A. Treuille, A. McNamara, Z. Popovic, J. Stam 2003, Keyframe control of smoke simulations, *ACM Transactions on Graphics 22*, 3, pp. 716-723

[3] R. Fattal, D. Lischinski 2004, Target-driven smoke animation, *ACM Transactions on Graphics 23*, 3, pp. 439-446

[4] M. B. Nielsen, B. B. Christensen 2010, Improved variational guiding of smoke animations, *Computer Graphics Forum 29*, 2, pp. 705-712

[5] Z. Yuan, F. Chen, Y. Zhao 2011, Pattern-guided smoke animation with lagrangian coherent structure, *ACM Transactions on Graphics 30*, 6, Article 136

[6] T. Kim, N. Thurey, D. James, M. Gross 2008, Wavelet turbulence for fluid simulation, *ACM Transactions on Graphics 27*, 3, Article 3

[7] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery 2007, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, Cambridge University Press

[8] J. Stam 1999, Stable fluids, In *Proceedings of ACM SIGGRAPH 1999, Annual Conference Series*, 121-128

[9] R. Bridson 2008, *Fluid Simulation for Computer Graphics*, AK Peters

[10] N. Max, R. Crawfis, D. Williams 1992, Visualizing wind velocities by advecting cloud textures, In *Proceedings of Visualization'92*, 179-183

[11] Y. Dobashi, K. Kusumoto, T. Nishita, T. Yamamoto 2008, Feedback control of cumuliform cloud formation based on computational fluid dynamics, *ACM Transactions on Graphics 27*, 3, Article 94